Supplementary Material: Copula-HDP-HMM: Non-parametric models with Copulas for I/O Efficient Bulk Cache Preloading

Appendix A: Brief Background: Bayesian Non-parametric Models

We now briefly discuss background on Bayesian Non-Parametric modeling. We define $\beta \sim GEM(\alpha)$ if $\beta_1 \sim Beta(1, \alpha)$ and $\forall k \geq 2$ $\beta_k = \eta_k \Pi_{p=1}^{k-1}(1 - \eta_p)$ $\eta_k \sim Beta(1, \alpha)$ A probability distribution is said to be sampled from a Dirichlet Process(DP) [26, 27] if $G \sim DP(\alpha, H)$

$$G = \sum_{k=1}^{\infty} \beta_k \delta_{\theta_k}, \beta \sim GEM(\alpha),\ \theta_k \sim H, k = 1 \ldots$$

where H is the base distribution of the DP.

Probability measures $G_1, \ldots, G_J$ follow **Hierarchical Dirichlet process** (HDP) [15] if

$$G_j \sim DP(\alpha, G_0), j = 1 \ldots J \ \text{ where } \ G_0 \sim DP(\alpha, H)$$

The HDP can be used as a prior for the transition probabilities in a Hidden Markov Model (HMM) to model a powerful extension of HMM that allows infinite number of states without having to fix the number of states in advance. The HDP-HMM [15] is defined as follows.

$$\beta \sim GEM(\gamma)$$
$$\pi_{\mathbf{k}}|\beta, \alpha_k \sim DP(\alpha_k, \beta), \theta_k|H \sim H, k = 1, 2, \ldots$$
$$Z_t|Z_t - 1, \pi \sim \pi_{Z_{t-1}}, X_t|Z_t \sim f_{Z_t}(\theta_k), t \in [T]$$

Appendix B: Inference Details

Sampling μ and Σ: Update Rules for the mean μ_k and the variance Σ_k random variables, $k \in [K]$, can be evaluated in closed form due to conjugacy of their prior distributions. The expression for their gibbs update for μ_k can be derived as the following where $\bar{\mu}_k$ is the sample mean of all observations assigned to cluster k.

$$\mu_k|\mathbf{Y}^k; \mu_0, \sigma_0 \sim \mathcal{N}(\hat{\mu}, \hat{\Sigma}) \tag{10.4}$$

where $\hat{\mu} = \Sigma_0(\Sigma_0 + \frac{1}{n_k}\Sigma_k)^{-1}(\bar{\mu}_k) + \frac{1}{n_k}\Sigma_k(\Sigma_0 + \frac{1}{n_k}\Sigma_k)^{-1}\mu_0$

$$\hat{\Sigma}_k = \frac{1}{n_k}\Sigma_0(\Sigma_0 + \frac{1}{n_k}\Sigma_k)^{-1}\Sigma_k$$

Σ_k can be sampled from the inverse Wishart posterior due to conjugacy.

$$\Sigma_k|\mathbf{Y}^k, \mu_k; S0, n0 \sim rwish(n_k\tilde{\Sigma}_k + S_0, n_k + n_0) \tag{10.5}$$

$$\text{where } \ \tilde{\Sigma}_k = \frac{1}{n_k}(\mathbf{Y}^k - \mu)^T(\mathbf{Y}^k - \mu)$$

Appendix C: Preliminaries: Cache Preloading

Storage systems comprise of a hierarchy of storage media. The Cache is a more expensive and faster storage medium (with access times in the order of microseconds), while the disk is a relatively inexpensive, slower storage medium (with access times in the order of milliseconds). When a task requests a set of blocks, if the blocks are in the cache they are served from the cache (hit), else they are served from disk and also put in cache for future use (miss). The hitrate, an important performance criterion is the fraction of hits out of all the requests.

Bulk Cache Preloading : Cache preloading is the process of predicting data that the application is likely to access in advance and loading it into cache. Bulk cache preloading refers to predicting large chunks of data, often in GB, well in advance, i.e for accesses that are scheduled to be requested several seconds, minutes or even hours in advance.

Appendix D: Experiment Details

10.1 Dataset Details: We use the dataset of publicly available Microsoft Server Storage Traces as that used in [25, 20] and an industrial trace. These traces are selected from a diverse set of workload types, such as SQL Server, Source control, Print Servers and so on as shown in table 5. We also generate synthetic traces with different characteristics to illustrate the performance of our algorithm[3]. The following Table 5 shows some details about our traces.

About 36 traces were originally present comprising about a week worth of data, hence allowing us to study long ranging temporal dependencies. We eliminated traces that are write heavy (write percentage > 25%) as we are focused on read cache. We present our results on the remaining traces. We also validated our results on one of our internal workloads, IN1 comprising data collected over 24 hours.

10.2 Simulator Design: We build a trace replay simulator for evaluating our Bulk Preloading strategy that replays a trace by reading each request in the live trace and servicing it. We build two versions of this

[3]Code for Copula-HDP-HMM and the baselines along with the synthetic datasets are available at `http://bit.ly/HulkSmashHitCache`

Table 5: Dataset Description

Acro -nym	Trace Name	Description
MS1	CAMRESWEBA03-lvm2	Web/SQL Srv
MS2	CAMRESSDPA03-lvm1	Source control
MS3	CAMUSP01-lvm1	Print Srv
MS4	CAM02SRV-lvm4	User Files
MS5	CAM02SRV-lvm3	Project Files
MS6	CAMRESWMSA03-lvm1	Media Server
IN1	NetApp InHouse	Industrial
SYN1	Synthetic Generated	Pure Sequential
SYN2	Synthetic Generated	Randomized Seq
SYN3	Synthetic Generated	Sea-Saw
SYN4	Synthetic Generated	Random Noisy Repeat

simulator.

- Simulator without Preload: This simulator on reading a request checks to see if the request is in cache. If so it is serviced from cache leading to a hit, else it constitutes a miss with the LRU replacement for evicting the cache. This is our baseline LRU simulator. This setting corresponds to the ∅-Preload strategy, that makes no predictions.

- Bulk Preload Simulator: In this version of the simulator, in addition to the LRU simulator, after every τ minutes (chosen slice length), the prediction algorithm is called. The predicted the set of accesses are loaded into cache and the trace replay continues until the next time slice. HULK our bulk preload strategy and the baseline strategies *Baseline-MVP-Preload* and *Baseline-IP-Preload* are evaluated with this simulator.

Experimental Setup: We use publicly available Block I/O traces, for our experiments that span about a day temporally. For the purpose of this experiment, we setup our experiment to use 50% of the available trace for the repository for observation and learning while we use the remaining 50% as the live trace on which we attempt preloading to evaluate caching performance. We note that in a real-world setting, we expect to have more trace data for learning the repository, that are collected at a prior time on the storage server. However, given the limited duration the publicly available traces, we use 50% of the available trace for training. We use a time slice duration of $\tau = 0.5 mins$ and we divide the available memory range into 100 bins. The cache size is fixed to about 5% of the total trace size. We note that for the datasize in this set up the prediction algorithm runs in less than 2 seconds, well within τ. We perform experiments to evaluate our bulk preloading strategy in terms of (1) Hitrates, (2) I/O efficiency in terms of Preload overhead

10.3 Data Simulation for Experiment 1

The Binomial, Neg-Binomial and the Poisson1 datasets are simulated from the generative model in sec 3 with the respective marginals while Poisson2 is generated from the generative model in [3]. Each model has 4 hidden states. For Binomial, Neg-Binomial and Poisson1 datasets, the mean, variance ,transition probabilities and the marginals used are given below.

```
Variance of each cluster:
sigma(1,:,:)=[13.50  13.12    0.10
                     0        5.09    0.31
                     0          0     0.85];


 sigma(2,:,:)=[9.56    0.30    19.00
              0.03    15.23     .01
              0.00    0.01    10.28];


 sigma(3,:,:)=[7.00   11.00  11.00
              0      7.10   10.00
              0       0     7.00];

 sigma(4,:,:)=[18.61    0       0
              0        2.0     0
              0        0     14.68];

Means of clusters:
 mu(1,:)=[100, 100, 2];
 mu(2,:)=[100, 2, 100];
 mu(3,:)=[2, 100, 100];
 mu(4,:)=[100, 10, 500];


 Transition Probabilities:
  tr=[0.1 0.7 0.1 0.1
     0.1 0.1 0.7 0.1
     0.1 0.1 0.1 0.7
     0.7 0.1 0.1 0.1];

Marginal parameters:
for Dataset Binomial for each dim j:
N_j=mu(z,j),p_j=0.4
//where z is the curresponding cluster index

for Dataset Negative Binomial  for each dim j:
N_j=mu(z,j),p_j=0.4
//where z is the curresponding cluster index

for Dataset Poisson1  for each dim j:
lambda=mu(z,j))
//where z is the curresponding cluster index
```

For the Poisson2 dataset, we use the following parameters of the multivariate Poisson [3]. Note that the transition probabilities are same as that of the previous dataset.

```
 lambda(1,:,:)=[150.32  1812.69    0.10
                     0    10.88    30.78
                     0       0     8.57];


lambda(2,:,:)=[56.0239    30.45    1900.01
                  3.45      172.31    1
                  0.01         1     28.04];

lambda(3,:,:)=[1000.254    1100.120   100.15
                      0       10.173     1000.00
                      0            0     0.02];

lambda(4,:,:)=[100.3410    1000    0
                        0     0      900
                        0     0      500.51];
```

Appendix E: Details of Prediction Algorithm

We describe details of the prediction algorithm (in algorithm 4) based on dynamic programming that aligns a sequences of state IDs of the live trace(recent history) with a sequence of state IDs in the repository.

The input to the prediction algorithm comprises of a sequence of state IDs for the live trace $\bar{\mathcal{Z}}_l = (\bar{Z}_1, \ldots, \bar{Z}_{T_{live}})$ a sequence of state IDs for the repository trace $\mathcal{Z}_R = (Z_1, \ldots, Z_{T_{repo}})$. Here T_{live} denotes the length of the live trace and T_{repo} denotes the length of the repository trace. The aim of the prediction algorithm is to find the best locally aligned subsequence of $\mathcal{Z}_L$, the live trace with the sequence $\mathcal{Z}_R$ (of the repo in its entirety). If the algorithm finds an alignment point, it returns a set $P \subset [T_{live}]$ of slice indices, containing a window of W slices in the repository around the alignment point that are the best candidates for preload.

We introduce a weightage function $wfunc(t, T) = 1 + \frac{t^2}{T}$ to assign positional weights. We also note the use of S in algorithm 3 that is a cluster similarity matrix. This can be derived as the distance between the means $\{\mu_k\}_{k=1}^{K}$ that are an additional outcome of the clustering step. The *Cluster Similarity Matrix* $S \in \mathcal{R}^{K\times K}$ is computed as follows : The similarity between two clusters k and l is obtained by the following simple reciprocal of distance, with a 1 added in denominator to prevent ∞ : $S[k,l] = (1/(1 + \|\mu_k - \mu_l\|^2))^{\frac{1}{2}}$ where μ_k and μ_l are the cluster means (Note the μ variables are discussed in more detail when they are introduced in section 3, describing our copula based clustering algorithm). (Note: We use $T_{live} = 50$ and W=5 in experiments).

Algorithm 4: **Prediction Algorithm** based on finding the best locally aligned subsequence of the repo sequence with the end of live sequence weighted by the position in the live sequence

```
Input   : $\bar{\mathcal{Z}}_L = \{\bar{Z}_1, \ldots, \bar{Z}_{T_{live}}\}$,$\mathcal{Z}_R = \{Z_1, \ldots, Z_{T_{repo}}\}$,
          $S \in \mathcal{R}^{K\times K}$
Output  : set $P \subset [T_{live}]$
$M[s,t] = 0, \forall s \in \{1, \ldots, T_{repo}\}, t \in \{1, \ldots, T_{live}\}$
for $s \in \{1, \ldots, T_{repo}\}$ do
    for $t \in \{1, \ldots, T_{live}\}$ do

                  $obj[1] = (M[s-1, t-1] +$
            $S(Z_{s-1}, \bar{Z}_{t-1}) * wfunc(t, N_R))$   Match
                  $obj[2] = M[s, t-1] - \delta$   Insert
                  $obj[3] = M[s-1, t] - \delta$   Delete
                  $obj[4] = 0$   Restart

        $M[s,t] = Max(obj)$
        Switch($argMax(obj)$)

            Case Match:     $Prev(s,t) = (s-1, t-1)$
            Case Insert:    $Prev(s,t) = (s, t-1)$
            Case Delete:    $Prev(s,t) = (s-1, t)$
            Case Restart:   $Prev(s,t) = (s, t)$

$\hat{t} = T_{live}$
$\hat{s} = ArgMax(M[:, T_{live}])$
return  : $P = \{s+1, \ldots, s+1+W\}$
```